Programación avanzada en Python

Lección 5: Trabajando con colecciones

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Trabajando con colecciones

PRESENTACIÓN Y OBJETIVOS

En este quinto capítulo vamos a estudiar como trabajar con colecciones. En Python principalmente tenemos 3 tipos de colecciones, listas, conjuntos y diccionarios. Veremos los principales métodos que podemos utilizar con estos tipos de colecciones.

Objetivos

- En esta lección aprenderás a:
- Las principales tipos de datos de colecciones.
- Trabajar con cadenas, listas, conjuntos y diccionarios.
- Diferenciar cada una de las colecciones.

1. CADENAS

Una cadena puede definirse en Python entre comillas simples y dobles. Una vez declara podemos imprimirla por pantalla con la función *print*

```
print("string")
```

```
string
```

Una cadena puede ser asignada a una variable y utilizarse para diferentes acciones.

```
variable = "string"
print(variable)
```

```
string
```

Si la cadena son sentencias que tienen más de una línea, podemos declararla con 3 comillas dobles o simples:

```
var = """hello
it is a string
yes"""
```

```
print(var)
```

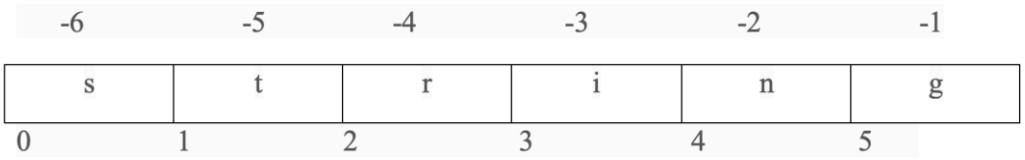
```
hello
it is a string
yes
```

1.1 Indexación de cadenas

Python no tiene un tipo de datos de caracteres, pero cada carácter de las cadenas puede ser asignado con longitud uno como un array.

El índice de la cadena comienza desde cero y se puede acceder a cada carácter especificando su valor de índice.

Sintaxis: nombre_de_la_cadena_variable [valor_del_índice]




```
var = "strong"
print(var[0])
print(var[1])
print(var[5])
```

```
s
t
g
```

La indexación negativa también es posible:

```
print(var[-1])
print(var[-6])
```

```
g
s
```

También es posible aplicar *slicing* a la cadena:

```
print(var[2:5])
```

```
ron
```

Podemos recorrer la cadena usando un bucle for:

```
for i in "string":
  print(i)
```

Aquí 'i' puede recorrer cada índice de la cadena y acceder al elemento particular.

Es posible comprobar la existencia de una cadena en particular con la palabra clave 'in' y retornar un valor booleano verdadero si está presente.

```
sentance = "Python is simple to learn"
print("simple" in sentance)
```

```
True
```

```
if "simple" in sentance:
  print("The word found")
```

```
The word found
```

Si la palabra no está presente en la frase, también se puede comprobar utilizando 'if not':

```
if "hello" not in sentance:
  print("word is not present")
```

```
word is not present
```

Operadores para cadenas

Los principales operadores utilizados para las cadenas son + y *.

Ya hemos hablado del operador +, que se utiliza para la concatenación de cadenas. El operador '*' se utiliza para repetir la palabra o cadena con el número de veces especificado.

Sintaxis: cadena * número de veces

```
a = "string.."
print(a * 3)
```

```
string..string..string..
```

Funciones de cadena

chr () - convierte un entero en carácter o su correspondiente valor ASCII

chr(97)	chr(33)
'a'	'!'

ord () - realiza la operación inversa a chr () y convierte el correspondiente valor entero o ASCII

ord("!")	ord("a")
33	97

str () - devuelve el parámetro especificado como una cadena. Si se comprueba el tipo de su valor devuelto, será una cadena independientemente del tipo del parámetro.

```
num = str(43)
print(num)
print(type(num))
```

```
43
<class 'str'>
```

Una cadena no puede ser modificada usando el operador de asignación de elementos '='

```
var = "cat"
var[0] = 'r'
```

```
TypeError: 'str' object does not support item assignment
```

En cambio, podemos utilizar la función *replace ()* para sustituir un carácter por otro.
Sintaxis: nombre_de_cadena.replace (char existente, char nuevo)

```
var.replace('c','r')
```

```
'rat'
```

2. LISTAS

La lista es un conjunto de elementos almacenados de forma ordenada.

Sintaxis:

Nombre_de_la_lista = [elemento1, elemento2, elemento 3... elemento n]

No hay límite para el número de elementos y éstos pueden ser de diferentes tipos de datos.

```
pet_list = ['dog','cat','rabbit']
```

Cada elemento tiene un índice, que empieza por cero y se puede acceder a los elementos por su posición de índice.

```
print(pet_list[2])
```

```
rabbit
```

2.1 Métodos de la lista

append ()

Se utiliza para añadir un solo elemento al final de la lista. Puede añadir fácilmente una cadena de números, tuplas y listas.

Veamos unos ejemplos:

Inicialmente definimos una lista vacía:

```
list = []
```

Añadimos elementos con la función append:

```
list.append(1)
list.append(2)
print(list)
```

```
[1, 2]
```

Las listas y tuplas también pueden añadirse como un solo elemento usando *append:*

```
list.append([3,4])
print(list)
```

```
[1, 2, [3, 4]]
```

```
list.append(('hello','hai'))
```

```
print(list)
```

```
[1, 2, [3, 4], ('hello', 'hai')]
```

Se pueden añadir múltiples elementos usando *append* con bucles

```
num_list = []
for i in range(5):
  num_list.append(i)

print(num_list)
```

```
[0, 1, 2, 3, 4]
```

insert ()

La función *insert* tiene el mismo propósito que *append* (), añadir elementos a la lista. La principal desventaja de append es que sólo añade elementos al final de la lista. La función insert puede añadir el elemento en la posición deseada según los parámetros especificados.

Sintaxis:

Nombre_de_la_lista. insert (posición, elemento)

Ejemplos:

```
num_list
```

```
[0, 1, 2, 3, 4]
```

Añadimos un elemento en la posición inicial :

```
num_list.insert(0,'a')
```

```
print(num_list)
```

```
['a', 0, 1, 2, 3, 4]
```

Añadimos un elemento entre 3 y 4:

```
num_list.insert(5,'b')
print(num_list)
```

```
['a', 0, 1, 2, 3, 'b', 4]
```

extend()

Esta función es la misma que insert y append utilizada para añadir elementos a la lista. Puede añadir varios elementos a la vez. Añade elementos al final de la lista como la función append y múltiples elementos como una lista de elementos.

Sintaxis:

List.extend([item1, item2,....item n])

```
list = [0, 1, 2, 3, 4]
list.extend([5,6,7,8,9,10])
print(list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

remove ()

La función remove elimina los elementos de la lista. Elimina un elemento a la vez especificando dicho elemento por parámetro. El elemento especificado será eliminado sin tener en cuenta su posición. Si el elemento no existe, se genera un mensaje de error.

De la lista anterior de 0 a 10, eliminamos el número 5:

```
list.remove(5)

print(list)
```

```
[0, 1, 2, 3, 4, 6, 7, 8, 9, 10]
```

Para eliminar varios elementos, podemos utilizar bucles:

```
for i in range(6,10):
  list.remove(i)

print(list)
```

```
[0, 1, 2, 3, 4, 10]
```

pop ()

La función pop se utiliza para eliminar los elementos de la lista. La función puede ser utilizada con o sin parámetro. Por defecto elimina el elemento del final de la lista. También puede eliminar un elemento especificado como parámetro a la función pop.

Vamos a comprobar ambos casos:

Si no se especifica ningún parámetro, se elimina el último elemento 6:

```
list = [0, 1, 2, 3, 4,5,6]
list.pop()
print(list)
```

```
[0, 1, 2, 3, 4, 5]
```

Cuando especificamos el elemento como parámetro, elimina el elemento especificado de la lista:

```
list.pop(4)
print(list)
```

```
[0, 1, 2, 3, 5]
```

clear ()

Elimina todos los elementos de la lista.

```
list = [0, 1, 2, 3, 4,5,6]
list.clear()
print(list)
```

```
[]
```

index()

```
list = [1, 2, 3, 4,5,6]
index = list.index(2)
print(index)
```

```
1
```

Devuelve el índice del elemento especificado en el parámetro.

count()

Devuelve el número de veces que se repite el elemento pasado por parámetro.

```
list = [1,2,3,3,4,3,5,3]
count = list.count(3)
print(count)
```

```
4
```

sort()

Ordena los elementos de la lista en orden ascendente por defecto:

```
list = [3,1,5,4,6,2]
list.sort()
print(list)
```

```
[1, 2, 3, 4, 5, 6]
```

Otro ejemplo con elementos de cadena

```
list = ['def','cds','abc','xyz','pqr']
list.sort()
print(list)
```

```
['abc', 'cds', 'def', 'pqr', 'xyz']
```

Hay dos parámetros para la función sort:

1. Reverse = True/False: puede especificar el parámetro reverse como true o false. Si es verdadero, entonces ordena en orden descendente.

```
list = [3,1,5,4,6,2]
list.sort(reverse=True)
print(list)
```

```
[6, 5, 4, 3, 2, 1]
```

2. Key = función: en la función puede especificar la condición para ordenar

```
def Func(e):
  return len(e)

list = ['def','cdsd','abczz','x','qr']
list.sort(key=Func)
print(list)
```

```
['x', 'qr', 'def', 'cdsd', 'abczz']
```

En la función especificada al parámetro clave se pide ordenar los elementos en base a la longitud de los mismos.

reverse()

Invierte el orden de la lista

```
list = [1, 2, 3, 4,5,6]
list.reverse()
print(list)
```

```
[6, 5, 4, 3, 2, 1]
```

copy()

Devuelve una copia de una lista y no utiliza ningún parámetro.

```
list = [1,22,333,4444]
list1 = list.copy()
print(list,list1)
```

```
[1, 22, 333, 4444] [1, 22, 333, 4444]
```

sum()

Devuelve la suma de todos los elementos de la lista

```
list = [1, 2, 3, 4,5,6]
sum = sum(list)
print(sum)
```

```
21
```

min()

Devuelve el elemento mínimo o más pequeño de la lista

```
list = [1, 2, 3, 4,5,6]
sum = min(list)
print(sum)
```

```
1
```

max ()

Devuelve el elemento máximo o más grande de la lista

```
list = [1, 2, 3, 4,5,6]
sum = max(list)
print(sum)
```

```
6
```

3. CONJUNTOS

Los conjuntos son otra estructura de datos, que recogen los elementos dentro de llaves '{}'. La principal característica de los conjuntos es que son desordenados y la indexación no funciona para este tipo de datos.

Sintaxis:

nombre_conjunto = {elemento 1, elemento 2... elemento n}

Ejemplo:

```
country = {"India","USA","UAE"}
```

Vamos a comprobar el tipo de estos datos:

```
print(type(country))
```

```
<class 'set'>
```

add()

Los conjuntos pueden añadir elementos utilizando la función *add*. Se añade un único elemento en cualquier lugar.

```
set = {1,2,4}
set.add(3)
print(set)
```

```
{1, 2, 3, 4}
```

update()

Con esta función también es posible añadir elementos en el conjunto. De esta manera se pueden añadir múltiples elementos en el parámetro y deben ser tuplas, listas o cadenas.

```
set = {1,2,4}
set.update([5,6],{7,8})
print(set)
```

```
{1, 2, 4, 5, 6, 7, 8}
```

discard()

Se utiliza para eliminar un elemento del conjunto. Elimina un solo elemento a la vez.

```
set = {1,2,4}
set.discard(4)
print(set)
```

```
{1, 2}
```

Las funciones remove(), pop() y clear () que trabajan para set son las mismas que las de list que ya hemos discutido.

3.1 Operaciones con conjuntos

union()

La unión de dos conjuntos es la combinación de ambos conjuntos. Se puede representar usando el operador '|' o con la función *union()*

```
set1 = {2,3,4}
set2 = {5,6,7}
print(set1 | set2)
```

```
{2, 3, 4, 5, 6, 7}
```

O con la función

```
print(set1.union(set2))
```

```
{2, 3, 4, 5, 6, 7}
```

intersection()

La intersección de dos conjuntos contiene los elementos comunes que está presente en ambas listas.

Se puede especificar con '&' o utilizando la función intersección ()

```
set1 = {2,3,4}
set2 = {5,3,2}
print(set1 & set2)
```

```
{2, 3}
```

```
print(set1.intersection(set2))
```

```
{2, 3}
```

difference()

La diferencia entre dos conjuntos significa, conjunto1 - conjunto2, contiene los elementos del conjunto1 pero no del conjunto2.

```
set1 = {2,3,4}
set2 = {5,3,2}
print(set1 - set2)
```

```
{4}
```

4. DICCIONARIOS

Los diccionarios son otro tipo de estructura de datos. Aquí cada elemento del diccionario tiene 2 parámetros: clave y valor. Así que cada elemento del diccionario es un par clave-valor separado por dos puntos. A cada valor le corresponde una clave.

Sintaxis:

diccionario_nombre : { clave1: valor1,

clave2:valor2,

..... }

Cada par clave-valor está separado por una coma. Los elementos están encerrados en llaves.

Ejemplo:

```
emails = { "xyz" : "xyz@gmail.com",
           "qas" : "qas@yahoo.com",
           "qwe" : "qwe@gmail.com"}
```

get ()

Se utiliza para acceder al valor con la clave como argumento.

```
dict = {'name':'abc',
        'age': 44}

print(dict.get('name'))
```

```
abc
```

pop ()

Se utiliza para eliminar un valor del diccionario dando la clave como parámetro.

```
dict = {'name':'abc',
        'age': 44}

dict.pop('name')
print(dict)
```

```
{'age': 44}
```

popitem ()

Se utiliza para eliminar un par clave-valor del diccionario. Aquí no se puede especificar ningún par en particular. Elimina arbitrariamente.

```
dict = {'name':'abc',
        'age': 44}

dict.popitem()
print(dict)
```

```
{'name': 'abc'}
```

clear()

Elimina todos los elementos del diccionario

```
dict = {'name':'abc',
        'age': 44}

dict.clear()
print(dict)
```

```
{}
```

5. PUNTOS CLAVE

- El índice de la **cadena** comienza desde cero y se puede acceder a cada carácter especificando su valor de índice.
- La **lista** es un conjunto de elementos almacenados de forma ordenada.
- La principal característica de los **conjuntos** es que son desordenados y la indexación no funciona para este tipo de datos.
- Cada elemento de un **diccionario** es un par clave-valor separado por dos puntos.